# On optimal and balanced sparse matrix partitioning problems

Anael Grandjean, Johannes Langguth, Bora Uçar

# On optimal and balanced sparse matrix partitioning problems

Anael Grandjean
LIP, ENS Lyon
46, allée d'Italie, ENS Lyon,
Lyon F-69364, France
Email:anael.grandjean@ens-lyon.fr

Johannes Langguth
LIP, ENS Lyon
46, allée d'Italie, ENS Lyon,
Lyon F-69364, France
Email:johannes.langguth@ens-lyon.fr

Bora Uçar
CNRS and LIP, ENS Lyon
46, allée d'Italie, ENS Lyon,
Lyon F-69364, France
Email: bora.ucar@ens-lyon.fr

***Abstract*—We investigate one dimensional partitioning of sparse matrices under a given ordering of the rows/columns. The partitioning constraint is to have load balance across processors when different parts are assigned to different processors. The load is defined as the number of rows, or columns, or the nonzeros assigned to a processor. The partitioning objective is to optimize different functions, including the well-known total communication volume arising in a distributed memory implementation of parallel sparse matrix-vector multiplication operations. The difference between our problem in this work and the general sparse matrix partitioning problem is that the parts should correspond to disjoint intervals of the given order. Whereas the partitioning problem without the interval constraint corresponds to the NP-complete hypergraph partitioning problem, the restricted problem corresponds to a polynomial-time solvable variant of the hypergraph partitioning problem. We adapt an existing dynamic programming algorithm designed for graphs to solve two related partitioning problems in graphs. We then propose graph models for a given hypergraph and a partitioning objective function so that the standard cutsize definition in the graph model exactly corresponds to the hypergraph partitioning objective function. In extensive experiments, we show that our proposed algorithm is helpful in practice. It even demonstrates performance superior to the standard hypergraph partitioners when the number of parts is high.**

## I. Introduction

Sparse matrix partitioning is an important problem arising in many applications. The applications include, among many others, sparse matrix ordering [8], parallelization of sparse matrix vector multiplication operations for distributed [7], [12] and shared memory systems [9], and crypto-system analysis [4]. All these applications, although different in nature, have two distinct partitioning constraint and three distinct partitioning objective functions. Hypergraph models have proved to be convenient abstractions in formulating the partitioning problems (see for example [10]). In particular, one associates a hypergraph model with a given sparse matrix and invokes standard hypergraph partitioning heuristics (available in many software libraries hMeTiS [17], MLpart [6], Mondriaan [28], Parkway [25], PaToH [13], and Zoltan [5]) to effectively tackle the partitioning problem.

In the standard one dimensional (1D) partitioning scheme of sparse matrices, one partitions either the rows or the columns of the matrix. Without loss of generality, we consider only the rowwise partitioning scheme—the columnwise partitioning can be seen as the rowwise partitioning of the transpose of a matrix. In this scheme, the partitioning constraint is have roughly equally sized parts in terms of the number of rows or the number of nonzeros. There are three standard objective functions which are defined in terms of *coupling columns*—those columns which have nonzeros in more than one parts. The first objective function is to minimize the number of coupling columns. The second objective function is to minimize the sum of the number of coupling columns each part touches. The third one is the difference of the first two. These standard objective functions are known as the *cut-net*, *soed*, and *connectivity-1* metrics in hypergraph partitioning domain. The standard 1D sparse matrix partitioning problem which is equivalent to the hypergraph partitioning problem is NP-hard.

In this work, we consider a variant of the standard 1D partitioning problem, called order restricted 1D partitioning problem. In this variant, the order of the rows is fixed. Apart from minimizing one of the three partitioning objective functions and meeting the partitioning constraint mentioned above, one has to obtain contiguous partitions of the rows in the given order. Formally, the additional restriction is that if rows $i$ and $j$ where $i < j$ are in a common part, then all rows between $i$ and $j$ are also in the same part. We have been motivated by PSPIKE family [21] of linear system solvers. As outlined in a recent study [24], a parallelization approach for these solvers is to first reorder the matrix in such a way that an important quantity of the matrix nonzeros are clustered around the main diagonal and then to partition the rows of the matrix sequentially. The aim of such partitioning is to have balance on parts and to minimize the coupling between parts (a formulation of which corresponds to the total communication volume). It is known [24, p. 108] that if one ignores the second objective, the partitioning problem can be solved with a known method [23]. The formulated partitioning problem encapsulates all of the partitioning requirements of PSPIKE family of solvers. A similar partitioning problem can be formulated for parallel direct solvers for sparse matrices where one first reorders the matrix and then partitions it among the available processors. Apart from these applications, we have two more general cases where such a partitioning

scheme can be useful in which we exploit the equivalence of hypergraph and sparse matrix partitioning problems. It is our experience that in partitioning a hypergraph into many parts of small size, the standard partitioning routines are not very effective. In such cases, a reasonable ordering of the vertices followed by an order restricted partitioning approach can be effective. Furthermore, the methods for the restricted partitioning problem can be easier to implement and use in applications where they can also be faster than the methods for the standard partitioning problem.

We formulate the order restricted 1D sparse matrix partitioning problem using hypergraph models which encode the three standard partitioning functions exactly. We show that the restricted problem is polynomial time solvable. In order to do so, we develop graph models whose cut properties corresponds exactly to the cut properties of the hypergraphs under the restricted partitioning formulation. We then adapt an earlier algorithm [19] to solve the restricted 1D sparse matrix partitioning problem under various formulations. The proposed algorithm and the models are interesting for hypergraph partitioning domain, since there is no graph based cut models for hypergraphs in general. Furthermore, previously known algorithms on restricted hypergraph partitioning problem [2] cannot handle two of the three objective functions described above.

The organization of the rest of this paper is as follows. In Section II, we formally define the restricted partitioning problem with various formulations and mention some related work. In Section III we present the proposed cut models which enable the use of graph partitioning algorithms. Section IV contains experimental investigations and Section V concludes the paper.

## II. Background, problem definition, and related work

### A. Hypergraphs and hypergraph partitioning

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ consists of a set of vertices $\mathcal{V}$ and a set of nets (hyperedges) $\mathcal{N}$. Every net is a subset of vertices. Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ is called a $K$A *-way partition* of the vertex set $\mathcal{V}$ if each part is non-empty, parts are pairwise disjoint i.e., $\mathcal{V}_k \cap \mathcal{V}_\ell = \emptyset$ for all $1 \leq k < \ell \leq K$, and the union of parts gives $\mathcal{V}$. Vertices can be associated with weights, denoted with $\mathtt{w}[\cdot]$, and nets can be associated with costs, denoted with $\mathtt{c}[\cdot]$.

Let $W_k$ denote the total weight in $\mathcal{V}_k$ (i.e., $W_k = \sum_{v \in \mathcal{V}_k} \mathtt{w}[v]$) and $W_{avg}$ denote the weight of each part when the total vertex weight is equally distributed (i.e., $W_{avg} = (\sum_{v \in \mathcal{V}} \mathtt{w}[v])/K$). If each part $\mathcal{V}_k \in \Pi$ satisfies the *balance criterion*

$$W_k \leq W_{avg}(1 + \varepsilon), \quad \text{for } k = 1, 2, \ldots, K \tag{1}$$

we say that $\Pi$ is *balanced* where $\varepsilon$ represents the maximum allowed imbalance ratio.

In a partition $\Pi$ of $\mathcal{H}$, a net that contains at least one vertex in a part is said to connect that part. *Connectivity* $\lambda_j$ of a net $n_j$ denotes the number of parts connected by $n_j$. There are three common objective functions

$$\chi_{cut}(\Pi) = \sum_{n \in \mathcal{N}_E} \mathtt{c}[n]\,, \tag{2}$$

$$\chi_{conn}(\Pi) = \sum_{n \in \mathcal{N}} \mathtt{c}[n](\lambda_n - 1)\,, \tag{3}$$

$$\chi_{SOED}(\Pi) = \sum_{n \in \mathcal{N}_E} \mathtt{c}[n]\lambda_n\,. \tag{4}$$

The cut-net metric given in (2) will be referred to here as *cut-net* metric, the one in (3) will be referred as *connectivity*-1 metric, and the one in (4) will be referred to as the SOED metric (widely used in the VLSI domain [1, p.10], [18], and recently found applications in the scientific computing domain [29]). Given $\varepsilon$ and an integer $K > 1$, the standard hypergraph partitioning problem can be defined as the task of finding a balanced partition $\Pi$ with $K$ parts such that $\chi(\Pi)$ is minimized. The standard hypergraph partitioning problem is NP-hard [20] with any of the above objective functions.

In the *column-net hypergraph model* [11], [12] $\mathcal{H}_\mathcal{R} = (\mathcal{V}_\mathcal{R}, \mathcal{N}_\mathcal{C})$ of an $M \times N$ sparse matrix $\mathbf{A}$, there exist one vertex $v_i \in \mathcal{V}_\mathcal{R}$ and one net $n_j \in \mathcal{N}_\mathcal{C}$ for each row $r_i$ and column $c_j$, respectively. Net $n_j \subseteq \mathcal{V}_\mathcal{R}$ contains the vertices corresponding to the rows that have a nonzero entry in column $c_j$. That is, $v_i \in n_j$ if and only if $a_{ij} \neq 0$. Weight $w_i$ of a vertex $v_i \in \mathcal{V}_\mathcal{R}$ is set to the total number of nonzeros in row $r_i$. The column-net model is used for rowwise partitioning.

### B. Problem definition

Let $A$ be a matrix. We generally assume that $A$ is sparse. The sequential partitioning problem asks for a partitioning of the rows of $A$ such that every part consists of a consecutive block of rows. Every row has a weight. The weight of a block of rows is equal to the sum of the weights of the individual rows residing therein. The weight of a row can be one, or the number of nonzeros in that row. We consider two variants of the problem. In the first one, we are given an upper bound and lower bound on the part sizes and each part should have weight between these bounds. We use the notation $\mathcal{P}(A, L, U)$ to denote an instance of this variant of the problem. In the second variant, we are also given the number $K$ of parts. In this case, one should obtain $K$ parts, each of which having weight between the lower and upper bounds. We use the notation $\mathcal{P}(A, L, U, K)$ to denote an instance of the second variant of the problem. In general, we are interested in finding partitionings that are optimal according to some metric. We consider three metrics explicitly in this paper. These metrics are (i) the total volume of communication; (ii) the number columns necessitating communication during parallel sparse matrix vector multiply operations; (iii) the some of the two.

Using the column-net hypergraph model, we obtain a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ from $A$. Since vertices in $\mathcal{H}$ correspond to rows in $A$, the row order of $A$ implies a fixed ordering $O$ of the vertices in $\mathcal{V}$. Let $O(v)$ be the unique ordinal number assigned to $v \in \mathcal{V}$ in $O$. We use $x \leq y$ to denote that the vertex $x$ is ordered before vertex $y$ in the given order $O$. Clearly, for

any partitioning of $\mathcal{V}$, we can apply the three metrics (2), (3), and (4) described above to compute a cutsize. Given this correspondence we use matrix and hypergraph interchangeably and apply the three cutsize metrics to a rowwise partitioning of matrix as well.

Thus, we obtain three variants of the problem of finding an optimal sequential partitioning of a matrix: Given $A$, $L$, $U$, and possibly $K$, find a ($K$-way if $K$ is given) partitioning of consecutive rows that is minimum according to the cut-net, connectivity-1, or SOED metrics. Every part must have a vertex weight between $L$ and $U$. Every part can be described as an interval between two vertices. The boundaries of such intervals will be called breakpoints.

In this paper, we will give algorithms for the weighted version of the problem, which has $w : \mathcal{V} \rightarrow \mathbb{N}$ as an additional input. The size of a partition is then given by the sum of the weights of the vertices contained in its interval. We define $d_{ij}$ as the *distance* i.e. the size of the interval between two vertices $v_i$ and $v_j$ for $i < j$ by $d_{ij} = \sum_{i \leq x \leq j} \mathsf{w}[x]$. Clearly, $d_{1M}$ denotes the total weight of $\mathcal{V}$.

### C. Related work

Pinar and Aykanat [22] investigate a similar partitioning problem for the 1D partitioning of sparse matrices. In their problem, the rows are again ordered. Given an integer $K \geq 2$, they find $K$ intervals of rows indices, each to be associated with a processor in such a way that the maximum load of a processor is minimum. They assume that the communication volume metric is handled implicitly. In order to highlight the difference with the considered paper, we give the following scenario. One can first compute the maximum load of a processor by the method of Pinar and Aykanat, and then use that number as an input to the algorithms that solve the sequential partitioning problem of this paper. This way one can obtain a partition that is optimal with respect to the maximum load of a processor and one of the cut size metrics.

Kernighan [19] investigates an optimal sequential partitioning problem in graphs. In this problem, we are given a vertex ordered graph and an upper bound on the part sizes. The aim is to obtain intervals of vertices, each defining a part such that part weights are less than the upper bound and the total weight of the edges that are cut by the partition is minimized. Kernighan proposes a dynamic programming algorithm to solve this problem optimally. The running time of the algorithm is $\mathcal{O}(UV)$ where $U$ is the given upper bound on part weights and $V$ is the number of vertices in the graph. We will describe this algorithm later, when we make use of it to solve the partitioning problems considered in this paper.

Alpert and Kahng [2] investigates an optimal sequential partitioning problem in hypergraphs for certain form of objective functions. If we ignore the restriction on the objective functions, the problem considered is $\mathcal{P}(A, L, U, K)$ with unit vertex weights. They propose a dynamic programming algorithm to solve the problem. Their dynamic programming approach necessitates a storage space of $M^2$ for a matrix with $M$ rows. The running time of the algorithm is $\mathcal{O}(K(U - L)M)$. The restriction on the objective function is that the total cutsize should be written in terms the contributions of each part. For example, given a partitioning $\Pi$, if one computes the number of cut nets that are connecting a part, and adds those numbers, one can compute the SOED function (4). That is, one can use the dynamic programming algorithm of Alpert and Kahng to optimize SOED. However, one cannot express the cut-net (2) and connectivity-1 (3) metrics in this way, and therefore cannot use that algorithm. The storage space of $M^2$ is used in order to achieve the mentioned running time complexity. Alpert and Kahng's formulation can also be used to find cyclic partitions, in which partitions are slices of a cyclic order (which increases the running time to $\mathcal{O}(K(U - L)M^2)$.

## III. Algorithms

Our strategy for finding optimal sequential partitions of hypergraphs consists of finding a reduction to the problem of finding optimal sequential partitions of graphs, and make use of a dynamic programming based algorithm proposed by Kernighan [19]. We will use a modified and extended version of this algorithm.

In order to find a suitable reduction, we need to find graphs that are *cut models* for hypergraphs under the given objective functions in this setting. In the general setting, this is only possible by introducing dummy vertices and negative edge weights, as shown by Ihler et al. [16]. However, having negative edge weights in a minimum cut partitioning problem, due to its relation to maximum cut problem creates difficulties. We show that due to fixed vertex ordering, it is possible to derive graphs with nonnegative edge weights that are cut models for hypergraphs under all three objective functions described in Section II. Our cut models are exact for hypergraphs with net costs. We therefore discuss the models with those costs but remind the reader that the matrix partitioning formulation, the nets have unit costs.

### A. Cut Models

The **cut-net** metric (2) allows for a relatively simple construction of cut model graphs. Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, we obtain the corresponding graph $G_{\mathcal{H}} = (\mathcal{V}, E)$ in the following manner. For every net $n \in \mathcal{N}$, $E$ contains exactly one edge $(v_{\min(n)}, v_{\max(n)})$ where $v_{\min(n)}$ is the vertex $v \in n$ having the minimum $O(v)$ among the vertices of $n$ and $v_{\max(n)}$ the vertex $v' \in n$ having the maximum $O(v')$ among the vertices of $n$. There are no other edges in $E$. The edge added for net $n$ bears a cost of $\mathsf{c}[n]$. In case of duplicate edges, the corresponding edge $e$ is assigned a cost $c_e$ equal to the sum of the costs of the associated nets. An example is given in Figure 1 for a hypergraph with unit net costs. A sample hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ where $\mathcal{V} = \{1, 2, 3, 4\}$ and $\mathcal{N} = \{h_1, h_2, h_3\}$ with $h_1 = \{1, 4\}, h_2 = \{1, 2, 4\}, h_3 = \{1, 2, 3\}\}$) on 4 vertices and 3 nets is shown on the left. The vertices and the nets of this hypergraph are shown with, respectively, numbered circles and ovals encompassing the vertices. The graph modeling the cut of this hypergraph under the cut-net metric (2) is shown on the right.

Now, if the vertex set $\mathcal{V}$ of $\mathcal{H}$ is partitioned such that any two vertices of a net $n$ are in different partitions, due to the fixed ordering $v_{\min(n)}$ and $v_{\max(n)}$ must be in different partitions of $\mathcal{H}$. In that case, $n$ adds exactly $\mathtt{c}[n]$ towards the cutsize definition of cut-net (2). In that case, applying the same partition to $\mathcal{V}$ of $G$ implies that the edge $e = (v_{\min(n)}, v_{\max(n)})$ corresponding to $n$ is cut, thereby increasing the cutsize in $G$ by $c_e = \mathtt{c}[n]$.

On the other hand, if $\mathcal{V}$ of $\mathcal{H}$ is partitioned such that $v_{\min(n)}$ and $v_{\max(n)}$ are in the same partition, all vertices in $n$ must be in the same partitions as well, and therefore, $n$ contributes 0 to the size of the cut. Now, for the same partitioning of $\mathcal{V}$ of $G$, the edge $e = (v_{\min(n)}, v_{\max(n)})$ is uncut since $v_{\min(n)}$ and $v_{\max(n)}$ are in the same partition and thus do not increase the cutsize in $G$.

Therefore, the proposed graph model, in which each net is represented as a single edge are a cut model for the corresponding hypergraphs under the cut-net metric.

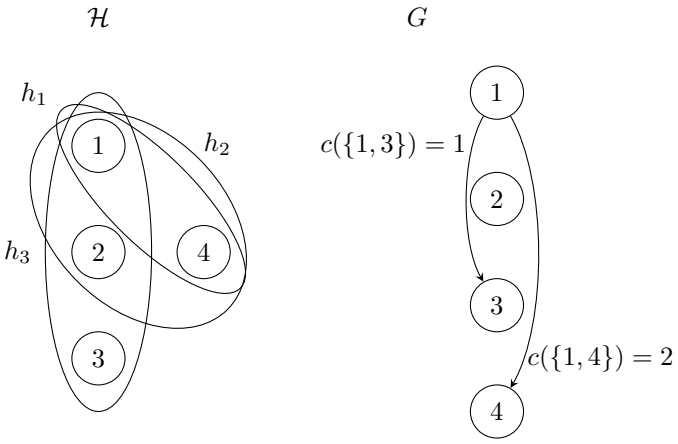



Fig. 1. *Cut-model graph $G$ for Hypergraph $\mathcal{H}$ under the cut-net metric, assuming nets with unit costs. For each hyperedge $h_j$, $G$ has an edge $e_j$ that connects the first vertex in the hyperedge to the last. All edges have a cost equal to the number of hyperedges $\mathcal{H}$ having the same first and last vertex.*

We use a similar approach for the **connectivity-1** metric (3). Given the hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, we construct the corresponding graph $G_{\mathcal{H}} = (\mathcal{V}, E)$ by adding $|n| - 1$ edges to $E$ for every net $n \in \mathcal{N}$. Let $n = \{v_1, v_2, \ldots, v_k\}$ be a hyperedge. Then the corresponding edges to be added to $E$ are $(v_1, v_2)$, $(v_2, v_3)$,..., $(v_k - 1, v_k)$. That is each net induces a chain on the vertices it contains. The weight of every edge on this chain is set to $\mathtt{c}[n]$. Again, there are no other edges in $E$. The cost $c_e$ of any edge $e$ is equivalent to sum of the costs of the associated nets. The cut model for the sample hypergraph of the previous figure for the connectivity-1 metric is shown in Figure 2.

Now, if the vertex set $\mathcal{V}$ of $\mathcal{H}$ is partitioned such that any two vertices $v_i$ and $v_{i+1}$ of a net $n$ are in different partitions, the objective function value increases by $\mathtt{c}[n]$. Of course, there can be up to $n - 1$ such vertex pairs in $n$, each increasing the objective function value by $\mathtt{c}[n]$, thereby measuring the total contribution of the net $n$ towards the cutsize definition of connectivity-1 metric (3). Clearly, for every such pair, one edge in $E$ is cut when applying the same partitioning of $\mathcal{V}$ to $G$ because every consecutive pair of vertices in a net is represented by one edge in $E$. Conversely, any cut edge in $E$ corresponds to a vertex pair $v_i$ and $v_{i+1}$ in the same net belonging to different partitions. Thus, the objective function value for a cut in the graph $G$ and in the hypergraph $\mathcal{H}$ are identical.

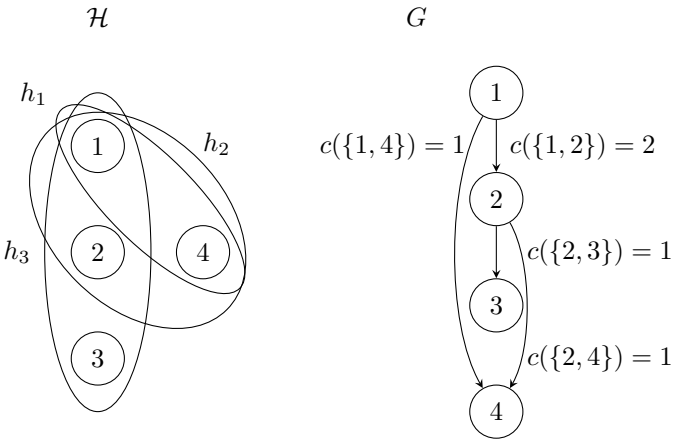



Fig. 2. *Cut-model graph $G$ for Hypergraph $\mathcal{H}$ under the connectivity-1 metric. Each edge in $G$ has a cost equal to the number of hyperedges of $\mathcal{H}$ that contain its consecutive vertex pair.*

Finally, we consider the SOED metric given by (4). It is essentially the sum of the objective functions of connectivity-1 (3) and cut-net 2. Therefore, we could easily compute its value on graphs by performing both transformations described above, thereby obtaining two graphs for which the objective function values can be added. That is, in this model one adds both types of edges to $E$. Let $n = \{v_1, v_2, \ldots, v_k\}$ be a net. Then, one adds the edge $(v_{\min(n)} = v_1, v_{\max(n)} = v_k)$ as in the model for the cut-net metric and and the edges $(v_1, v_2)$, $(v_2, v_3)$,..., $(v_k - 1, v_k)$ as in the connectivity-1 metric. In other words, each net $n = \{v_1, v_2, \ldots, v_k\}$ induces a cycle on its vertices in the graph model. The cut model for the sample hypergraph of the previous figure for the SOED metric is shown in Figure 3. One immediately verifies that the graph so obtained is a cut model under the SOED metric.

### *B. Kernighan's Dynamic Programming Algorithm*

An efficient algorithm for finding optimal sequential partitions of graphs was suggested by Kernighan [19]. The algorithm is shown in Figure 4. Given a vertex ordered graph $G$ with $M$ vertices, vertex weights, edge costs, and a maximum partition size $U$, it returns a sequential partitioning of $G$ having minimum cutsize. The output is encoded in breakpoints, i.e. the positions of the partition boundaries w.r.t. $O$.

The algorithm uses the dynamic programming paradigm introduced by Bellman [3]. It identifies the optimal substructure property that a breakpoint can be placed optimally at a vertex $x$, by examining optimal solutions at vertices $y < x$ for which $d_{y,x-1} \leq U$. Let $T(x)$ denote the cutsize of optimum

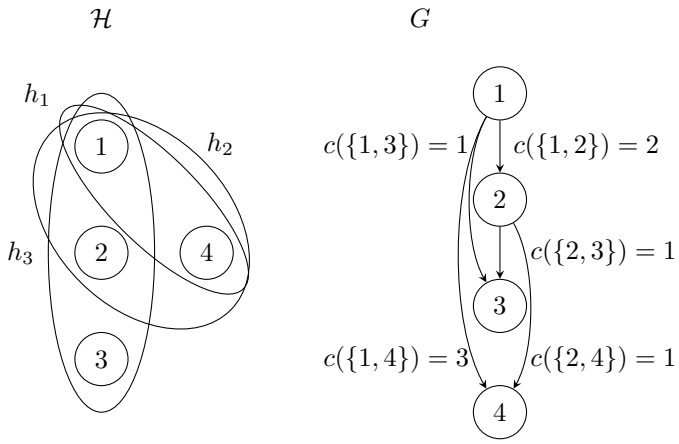



Fig. 3. *Cut-model graph $G$ for Hypergraph $\mathcal{H}$ under the SOED metric. The edge set of $G$ is the union of those for the connectivity-1 and cut-net graphs. In case of duplicate edges such as $\{1,4\}$, costs are summed up.*

**Input:** A weighted graph $G = (V, E)$ whose vertices are ordered, and an upper bound $U$

**Output:** A set of Breakpoints $z_1, z_2, \ldots, z_k$ in descending order.

1: $T(1) \leftarrow 0$
2: $R(1) \leftarrow 0$
3: **for** $x$ from 2 to $M + 1$ **do**
4: $\quad Y \leftarrow \{y | d_{y,x-1} \leq U\}$
5: $\quad T(x) \leftarrow \min_{y \in Y}(T(y) + C(x, y))$
6: $\quad R(x) \leftarrow \operatorname{argmin}_{y \in Y}(T(y) + C(x, y))$
$\quad\quad$ ▶Break ties by choosing minimum $y$
7: $z \leftarrow M + 1$
8: **while** $z > 1$ **do**
9: $\quad z \leftarrow R(z)$
10: $\quad$ OUTPUT $z$

Fig. 4. Kernighan's dynamic programming algorithm

partitioning of vertices $1, \ldots, x$ such that there is a break point at $x$, and let $R(x)$ denote the corresponding optimal previous breakpoint. Let $\psi(x)$ be the set of edges that are cut by the breakpoint at $x$. For a given set $F$ of edges, let $C(F)$ be the total sum of the cost of edges in $F$. In this setting, $C(\psi(x))$ is the cost of the edges cut by $x$ and $C(\psi(x) \setminus \psi(y))$ is the cost of the edges that are cut by $x$ but not cut by $y$. We use the shorthand notation $C(x, y)$ to denote $C(\psi(x) \setminus \psi(y))$. Thus:

$$C(x, y) = \sum_{y \leq i < x, j \geq x} c_{ij}$$

Then, due to the optimal substructure property, it must hold that $T(x) = \min_{y<x} T(y) + C(x, y)$ for $y$'s with $d_{y,x-1} \leq U$. Every time a $T(x)$ value is set, the smallest $y$ achieving $T(x)$ is stored as the breakpoint preceding the one at $x$. When the algorithm reaches to $x = M + 1$, we have thus $T(M + 1)$, the optimal partitioning of vertices $1, \ldots, M$ whose break points are set accordingly (the last one being at $x + 1$).

It is crucial to implement the algorithm in such a way that the values of $C(x, y)$ are not stored for all $x$ and $y$ and not recomputed from scratch every time they are needed. Kernighan [19, Theorem 4] discusses how to implement this by using $\mathcal{O}(M)$ space and by accessing every edge at most twice.

### C. The Modified Dynamic Programming Algorithm

We modify Kernighan's algorithm in two ways to be able to use it in solving the problems $\mathcal{P}(A, L, U)$ and $\mathcal{P}(A, L, U, K)$ for a given matrix $A$, the lower and upper bounds $L$ and $U$, and the number of parts $K$.

The first modification replaces the set of feasible preceding breakpoint locations $Y$ of a breakpoint at $x$. In this modification we set $Y = \{y | L \leq d_{y,x-1} \leq U\}$ at line 4 of the algorithm. Clearly, this will lead to a modified algorithm computing optimal partition size, but we need to adapt the initialization. In addition to setting $T(1) = 0$, we compute an $x_{\min}$ such that $L \leq d_{y,x_{\min}-1}$ and $L > d_{y,x_{\min}-2}$. We set $T(x_{\min}) = 0$ and for all $1 < x \leq x_{\min}$ we set $T(x)$ to infinity to denote infeasible breakpoints. With this modification, we can now solve problems of the form $\mathcal{P}(A, L, U)$ by first creating a suitable cut-model graph and then by applying the modified dynamic programming algorithm. The running time complexity of the algorithm remains as $\mathcal{O}(UM)$. With this modification, unless the values of $L$ and $U$ are very close, the number of partitions created varies so that we still cannot solve instances of the form $\mathcal{P}(A, L, U, K)$.

In order to solve $\mathcal{P}(A, L, U, K)$ problems, we need to modify the algorithm in such a way that it accepts an input variable $K$ and generates exactly $K$ partitions of size between $U$ and $L$ or determines that such a partitioning is infeasible. We do so by adding a second dimension of size $K$ to the table of intermediate results. The resulting algorithm is shown in Figure 5. Thus $T(x, k)$ in Algorithm 5 denotes the value of the optimum partitioning of $1, \ldots, x-1$ using $k$ parts. Clearly, $T(x, k)$ should be set to $\min_{y<x} T(y, k-1) + C(x, y)$ for $y$ satisfying $\{L \leq d_{y,x-1} \leq U\}$. Now, the main algorithm needs to loop over an additional variable, i.e., $k$ in every iteration over the main **for** loop. When accessing values of $T$, in the computation of $T(x, k)$, we have to access $T(y, k-1)$ as we are adding a potential new partition. Due to the $L$ and $U$ constraints, most $k$-way partitionings are infeasible and thus most entries of $T$ will be infinity. When the algorithm terminates the entry of $T(M+1, K)$ shows whether the desired partitioning is feasible, and if so $R(M + 1, K)$ points to the last breakpoint.

The addition of the number of parts $K$ as program parameter increases the time complexity of the algorithm to $\mathcal{O}(KUM)$, although much of the $K$ can be saved by computing only feasible $T(x, k)$ would be feasible. Space complexity of the algorithm increases to $\mathcal{O}(MK)$.

## IV. Experiments

We perform a series of experiments in order to compare speed and quality of our proposed partitioning method to

**Input:** A weighted graph $G = (V, E)$ whose vertices are ordered, a number of parts $K \in \mathbb{N}$, and upper and lower bounds $L, U$ on part sizes

**Output:** A set of Breakpoints $z_1, z_2, \ldots, z_K$ in descending order.

Find $x_{\min}$ such that $L \leq d_{y,x_{\min}-1}$ and $L > d_{y,x_{\min}-2}$
$T(x, 0) \leftarrow \inf$ for all $x \leq M$
**for** $x$ from 0 to $x_{\min} - 1$ **do**
    $T(x, k) \leftarrow \inf$ for all $k \in 0, \ldots, K$
$T(x_{\min}, 1) \leftarrow d_{1,x_{\min}}$
$R(1, 0) \leftarrow 0$
**for** $x$ from $x_{\min}$ to $M + 1$ **do**
    **for** $k$ from 1 to $K$ **do**
        $Y \leftarrow \{y | L \leq d_{y,x-1} \leq U\}$
        $T(x, k) \leftarrow \min_{y \in Y}(T(y, k-1) + C(x, y))$
        $R(x, k) \leftarrow \operatorname{argmin}_{y \in Y}(T(y, k-1) + C(x, y))$
            ▶Break ties by choosing the minimum $y$
$z \leftarrow M + 1$
$k \leftarrow K$
**while** $z > 1$ **do**
    $z \leftarrow R(z, k)$
    $k \leftarrow k - 1$
    OUTPUT $z$

Fig. 5. The modified dynamic programming algorithm

established alternatives. To this end, we use matrices from the University of Florida Sparse Matrix Collection [15]. Of course, it is not the aim of these experiments to compare two directly competing algorithm. We rather seek to compare running time and solution quality of sequential partitioning to that of general partitioning, and investigate whether sequential partitioning can be viable if the problem calls for it.

For comparison, we use the PaToH [26] hypergraph partitioner. We refer the proposed dynamic programming based algorithm (with or without $K$) as sequential DP. We compare speed and quality of sequential DP and PaToH for all three metrics under a variety of partitioning parameters. In order to enable PaToH to work using the SOED metric, some custom code adjustments were necessary. Of course, PaToH and the proposed partitioning routine do not solve the same problem. That is why the comparisons should only be taken as indicators of how the vertex order restriction affects the solution quality, and how the running time of the proposed algorithm behaves in a perspective to the known methods.

All our algorithms are implemented in the C programming language. The codes are called via a Matlab interface. The same is true for PaToH [26], [27]. We compiled our codes with *mex* of Matlab using *gcc* version 4.4.2 with the optimization flag -O and ran the compiled codes on a machine with an Intel Xeon E5520 Quad Core computer running at 2.27 Ghz. The system runs Debian Linux 2.6.32, Matlab 7.13 and gcc 4.4.5.

Our test set contains 17 matrices having between 50000 and 100000 rows. The matrices and their properties are listed in Table I. We set $K$ to 64 and allow for a 1% load imbalance.

TABLE I
TEST INSTANCES FOR THE MAIN EXPERIMENTS. #NZ IS THE NUMBER OF NONZEROS. *mCD* AND *mRD* DENOTE THE MINIMUM COLUMN AND ROW DEGREE RESPECTIVELY, WHILE *MAXCD* AND *MAXRD* DENOTE THE MAXIMUM COLUMN AND ROW DEGREES. NOTE THAT THE NAME OF INSTANCE *mark3jac140sc* IS SHORTENED TO *mark3jac* IN THE TABLES.

| Matrix | #rows | #cols | #NZ | *mCD* | *MAXCD* | *mRD* | *MAXRD* |
|---|---|---|---|---|---|---|---|
| epb3 | 84617 | 84617 | 463625 | 3 | 7 | 3 | 6 |
| ct20stif | 52329 | 52329 | 2600295 | 2 | 207 | 2 | 207 |
| circuit_4 | 80209 | 80209 | 307604 | 1 | 8900 | 1 | 6750 |
| bayer01 | 57735 | 57735 | 275094 | 1 | 33 | 1 | 28 |
| gupta2 | 62064 | 62064 | 4248286 | 3 | 8413 | 3 | 8413 |
| bcircuit | 68902 | 68902 | 375558 | 2 | 34 | 2 | 34 |
| g7jac180 | 53370 | 53370 | 641290 | 1 | 98 | 1 | 141 |
| mark3jac | 64089 | 64089 | 376395 | 1 | 46 | 2 | 44 |
| lhr71 | 70304 | 70304 | 1494006 | 1 | 36 | 1 | 63 |
| finan512 | 74752 | 74752 | 596992 | 3 | 55 | 3 | 55 |
| nasasrb | 54870 | 54870 | 2677324 | 12 | 276 | 12 | 276 |
| lp_nug30 | 52260 | 379350 | 1567800 | 4 | 60 | 30 | 30 |
| cfd1 | 70656 | 70656 | 1825580 | 12 | 33 | 12 | 33 |
| shyy161 | 76480 | 76480 | 329762 | 1 | 6 | 1 | 6 |
| venkat01 | 62424 | 62424 | 1717792 | 16 | 44 | 16 | 44 |
| qa8fk | 66127 | 66127 | 1660579 | 8 | 27 | 8 | 27 |
| pkustk03 | 63336 | 63336 | 3130416 | 12 | 90 | 12 | 90 |

TABLE II
RUNNING TIME FOR BOTH ALGORITHMS AND ALL THREE METRICS ON THE TEST SET FOR $K = 64$ AND 1% LOAD IMBALANCE.

| Matrix | Sequential DP | | | PaToH | | |
|---|---|---|---|---|---|---|
| | Cut | Con-1 | SOED | Cut | Con-1 | SOED |
| epb3 | 16.07 | 15.37 | 15.37 | 0.72 | 0.72 | 0.72 |
| ct20stif | 2.50 | 5.56 | 2.52 | 4.42 | 3.92 | 4.44 |
| circuit_4 | 15.84 | 15.79 | 15.85 | 1.33 | 0.66 | 1.23 |
| bayer01 | 8.63 | 8.25 | 8.58 | 0.52 | 0.53 | 0.52 |
| gupta2 | 2.57 | 8.60 | 2.59 | 27.43 | 5.43 | 27.04 |
| bcircuit | 3.46 | 4.22 | 3.50 | 0.67 | 0.67 | 0.66 |
| g7jac180 | 6.64 | 6.49 | 6.42 | 1.41 | 1.03 | 1.44 |
| mark3jac140sc | 9.16 | 8.76 | 8.49 | 1.09 | 0.97 | 1.09 |
| lhr71 | 12.86 | 12.06 | 12.75 | 2.00 | 1.96 | 2.03 |
| finan512 | 14.00 | 14.30 | 13.99 | 1.40 | 1.42 | 1.42 |
| nasasrb | 7.14 | 5.16 | 5.59 | 4.24 | 3.96 | 4.23 |
| lp_nug30 | 9.06 | 9.94 | 9.34 | 6.70 | 4.54 | 6.66 |
| cfd1 | 4.09 | 1.69 | 2.59 | 3.60 | 3.24 | 3.61 |
| shyy161 | 13.46 | 11.72 | 12.35 | 0.53 | 0.52 | 0.54 |
| venkat01 | 3.64 | 9.28 | 3.57 | 2.01 | 1.94 | 2.00 |
| qa8fk | 4.03 | 1.33 | 2.17 | 3.16 | 2.77 | 3.16 |
| pkustk03 | 10.89 | 9.98 | 10.77 | 4.89 | 4.64 | 4.88 |
| Average | 8.47 | 8.73 | 8.03 | 3.89 | 2.29 | 3.86 |

The results are shown in Tables II and III.

We note that PaToH is approximately twice as fast as our algorithm. In addition, he obtained cut sizes for PaToH are significantly smaller for all metrics. The difference is approximately a factor of 2. Therefore, we conclude that at this number of partitions, the DP algorithm is strictly inferior to PaToH. Note that for one matrix in the test set, *gupta2*, PaToH takes very long running time, but computes high solution quality. Our algorithm did not show this behavior on any test instance.

### A. Increased Load Imbalance

We consider the effect of increasing the allowed load imbalance for both algorithms from 1% to 10%. We keep $K$ at 64. The results, which are given in Table IV show that the running time of our algorithm is affected only marginally. Interestingly, running time for the SOED metric is lower than in the 1% load balance case. Running times for PaToH increased slightly.

TABLE III
SOLUTION QUALITY FOR BOTH ALGORITHMS AND ALL THREE METRICS ON THE TEST SET FOR $K = 64$ AND $1\%$ LOAD IMBALANCE.

| Matrix | Sequential DP | | | PaToH | | |
|---|---|---|---|---|---|---|
| | Cut | Con-1 | SOED | Cut | Con-1 | SOED |
| epb3 | 25326 | 25326 | 50652 | 7210 | 6876 | 14154 |
| ct20stif | 63421 | 35790 | 99529 | 25230 | 19501 | 45333 |
| circuit_4 | 37093 | 32862 | 69957 | 28510 | 19519 | 50455 |
| bayer01 | 4192 | 4150 | 8344 | 2338 | 2123 | 4439 |
| gupta2 | 165258 | 61086 | 226398 | 98725 | 59763 | 168116 |
| bcircuit | 75661 | 50075 | 125806 | 2332 | 2423 | 4447 |
| g7jac180 | 87619 | 17194 | 104826 | 16857 | 11633 | 30459 |
| mark3jac | 29617 | 24346 | 53975 | 18899 | 14355 | 34500 |
| lhr71 | 9466 | 9214 | 18692 | 6035 | 6254 | 11798 |
| finan512 | 54528 | 46489 | 101018 | 11843 | 10411 | 22210 |
| nasasrb | 46961 | 40609 | 87771 | 23141 | 19832 | 42925 |
| lp_nug30 | 1133669 | 379350 | 1513019 | 535681 | 342706 | 924190 |
| cfd1 | 135594 | 65059 | 200812 | 36392 | 29066 | 66043 |
| shyy161 | 16747 | 16747 | 33494 | 5786 | 5594 | 11330 |
| venkat01 | 48148 | 33536 | 81752 | 12568 | 11512 | 24720 |
| qa8fk | 125252 | 65150 | 190541 | 38091 | 28644 | 64929 |
| pkustk03 | 55326 | 38052 | 93828 | 18576 | 15306 | 34788 |
| Average | 124346 | 55590 | 180024 | 52248 | 35619 | 91461 |

This is due to the fact that the number of possible solutions to be checked increases. The solution quality, given in Table V, improves substantially for the DP approach in the connectivity-1 and SOED metrics and marginally for PaToH. We therefore conclude that our algorithm is more suited for dealing with situations that allow large load imbalances.

TABLE IV
RUNNING TIME AND AVERAGE SOLUTION QUALITY FOR BOTH ALGORITHMS AND ALL THREE METRICS ON THE TEST SET FOR $K = 64$ AND $10\%$ LOAD IMBALANCE.

| Matrix | Sequential DP | | | PaToH | | |
|---|---|---|---|---|---|---|
| | Con-1 | Cut | SOED | Con-1 | Cut | SOED |
| epb3 | 18.80 | 18.12 | 18.82 | 0.89 | 0.88 | 0.89 |
| ct20stif | 2.12 | 5.94 | 2.16 | 4.78 | 4.27 | 4.77 |
| circuit_4 | 10.80 | 17.50 | 11.20 | 1.17 | 0.60 | 1.17 |
| bayer01 | 9.70 | 9.32 | 9.70 | 0.63 | 0.64 | 0.65 |
| gupta2 | 2.95 | 10.92 | 3.03 | 27.22 | 5.54 | 28.37 |
| bcircuit | 3.25 | 2.11 | 2.70 | 0.79 | 0.79 | 0.79 |
| g7jac180 | 1.99 | 1.56 | 2.02 | 1.68 | 1.23 | 1.64 |
| mark3jac140sc | 2.30 | 11.41 | 2.36 | 1.54 | 1.33 | 1.49 |
| lhr71 | 13.07 | 13.32 | 12.88 | 2.18 | 2.15 | 2.17 |
| finan512 | 15.90 | 15.35 | 15.85 | 1.85 | 1.84 | 1.82 |
| nasasrb | 8.83 | 7.71 | 8.01 | 4.57 | 4.26 | 4.58 |
| lp_nug30 | 7.33 | 10.66 | 7.68 | 9.16 | 5.95 | 8.82 |
| cfd1 | 4.97 | 5.15 | 4.80 | 4.48 | 3.99 | 4.43 |
| shyy161 | 15.71 | 10.15 | 10.78 | 0.65 | 0.64 | 0.65 |
| venkat01 | 11.86 | 1.88 | 2.72 | 2.32 | 2.21 | 2.31 |
| qa8fk | 3.00 | 1.74 | 2.65 | 4.02 | 3.38 | 4.01 |
| pkustk03 | 7.92 | 11.11 | 7.84 | 5.11 | 4.79 | 5.14 |
| Average | 8.26 | 9.06 | 7.37 | 4.30 | 2.62 | 4.34 |

### *B. Varying Number of Partitions*

We consider the behavior of the algorithms under various values of $K$. First, we reduce $K$ to 16. The results are given in Table VI. We immediately observe that our algorithm slows down dramatically with decreasing $K$ while PaToH speeds up significantly. Due to fewer partition boundaries, all the cutsizes are smaller than in the $K = 64$ case, but the general ratio between both algorithms is unchanged. Therefore, the DP algorithm should not be used when the desired number of partitions is low.

TABLE V
SOLUTION QUALITY FOR BOTH ALGORITHMS AND ALL THREE METRICS ON THE TEST SET FOR $K = 64$ AND $10\%$ LOAD IMBALANCE.

| Matrix | Sequential DP | | | PaToH | | |
|---|---|---|---|---|---|---|
| | Con-1 | Cut | SOED | Con-1 | Cut | SOED |
| epb3 | 17849 | 17849 | 35698 | 7104 | 6936 | 13986 |
| ct20stif | 55209 | 35073 | 90672 | 23997 | 18184 | 42528 |
| circuit_4 | 108826 | 51563 | 160405 | 27118 | 19820 | 49021 |
| bayer01 | 41041 | 39503 | 80565 | 1951 | 1860 | 3795 |
| gupta2 | 219959 | 62048 | 282008 | 100116 | 59590 | 157837 |
| bcircuit | 19507 | 17134 | 36676 | 1873 | 1803 | 3701 |
| g7jac180 | 55816 | 41559 | 97398 | 16525 | 10763 | 30441 |
| mark3jac140sc | 77746 | 55556 | 133365 | 18541 | 14499 | 33709 |
| lhr71 | 8476 | 8458 | 16946 | 5492 | 5301 | 11022 |
| finan512 | 124002 | 66682 | 190816 | 9411 | 9280 | 18965 |
| nasasrb | 42114 | 38745 | 81984 | 20799 | 17986 | 38738 |
| lp_nug30 | 425700 | 376963 | 802854 | 541067 | 344100 | 909012 |
| cfd1 | 102973 | 58967 | 162743 | 34540 | 27881 | 63840 |
| shyy161 | 12143 | 11947 | 24134 | 5651 | 5456 | 11150 |
| venkat01 | 71288 | 44224 | 115784 | 11620 | 11016 | 22856 |
| qa8fk | 120981 | 64169 | 185545 | 37087 | 27937 | 64666 |
| pkustk03 | 49638 | 37290 | 87456 | 16920 | 14448 | 30654 |
| Average | 91369 | 60455 | 152062 | 51754 | 35109 | 88584 |

TABLE VI
RUNNING TIME AND AVERAGE SOLUTION QUALITY FOR BOTH ALGORITHMS AND ALL THREE METRICS ON THE TEST SET FOR $K = 16$ AND $1\%$ LOAD IMBALANCE.

| Matrix | Sequential DP | | | PaToH | | |
|---|---|---|---|---|---|---|
| | Con-1 | Cut | SOED | Con-1 | Cut | SOED |
| epb3 | 63.99 | 64.19 | 65.26 | 0.46 | 0.47 | 0.46 |
| ct20stif | 18.95 | 24.82 | 18.69 | 2.96 | 2.77 | 2.95 |
| circuit_4 | 58.59 | 58.61 | 58.66 | 0.77 | 0.50 | 1.43 |
| bayer01 | 30.72 | 30.36 | 30.72 | 0.31 | 0.31 | 0.31 |
| gupta2 | 4.21 | 32.15 | 4.25 | 16.94 | 5.32 | 16.19 |
| bcircuit | 42.81 | 42.28 | 41.12 | 0.42 | 0.42 | 0.42 |
| g7jac180 | 25.68 | 25.46 | 25.42 | 0.98 | 0.78 | 0.99 |
| mark3jac140sc | 36.44 | 35.47 | 35.36 | 0.72 | 0.70 | 0.72 |
| lhr71 | 44.98 | 45.12 | 46.20 | 1.31 | 1.31 | 1.26 |
| finan512 | 51.10 | 51.05 | 51.10 | 0.85 | 0.85 | 0.84 |
| nasasrb | 28.00 | 26.46 | 26.76 | 2.86 | 2.85 | 2.86 |
| lp_nug30 | 9.62 | 28.14 | 9.89 | 5.41 | 4.22 | 5.52 |
| cfd1 | 36.97 | 12.10 | 12.61 | 2.31 | 2.24 | 2.35 |
| shyy161 | 52.46 | 50.84 | 51.46 | 0.31 | 0.31 | 0.32 |
| venkat01 | 36.18 | 34.89 | 35.32 | 1.31 | 1.30 | 1.32 |
| qa8fk | 33.56 | 24.01 | 24.40 | 2.11 | 1.92 | 2.07 |
| pkustk03 | 29.86 | 36.55 | 29.68 | 3.26 | 3.18 | 3.26 |
| Average | 35.54 | 36.62 | 33.35 | 2.55 | 1.73 | 2.55 |
| Avg. quality | 93281 | 43333 | 136633 | 33846 | 27201 | 63980 |

Conversely, Table VII shows what happens when $K$ is raised to 256. Our algorithm works noticeably faster, especially for the connectivity-1 and SOED metrics where PaToH takes almost twice as long. This illustrates that the proposed DP approach is much more suitable for partitioning problems where the desired number of partitions is large. As expected, the solution quality is significantly lower in all cases, as shown in Table VIII. The difference between PaToH and our algorithm is smaller than in the $K = 64$ case, but remains significant.

We have seen that the DP algorithm is suited for creating a large number of small partitions. We performed an additional experiment using the variant of our algorithm described in Section III-C where $K$ was not fixed. Partition size was bound by $L = 50$ and $U = 100$. For comparison, PaToH was initialized with $K$ suitable for obtaining an average partition size of 75 and a load imbalance of $33\%$. The results in Table

TABLE VII
RUNNING TIME FOR BOTH ALGORITHMS AND ALL THREE METRICS ON THE TEST SET FOR $K = 256$ AND $1\%$ LOAD IMBALANCE.

| Matrix | Sequential DP | | | PaToH | | |
|---|---|---|---|---|---|---|
| | Con-1 | Cut | SOED | Con-1 | Cut | SOED |
| epb3 | 3.97 | 3.25 | 3.99 | 1.10 | 1.08 | 1.09 |
| ct20stif | 1.58 | 0.73 | 1.59 | 5.76 | 4.76 | 5.81 |
| circuit_4 | 2.09 | 1.66 | 2.15 | 1.84 | 0.79 | 1.70 |
| bayer01 | 1.15 | 0.75 | 1.18 | 0.80 | 0.76 | 0.80 |
| gupta2 | 2.15 | 3.06 | 2.35 | 33.61 | 5.78 | 34.97 |
| bcircuit | 1.51 | 0.94 | 1.53 | 1.00 | 0.98 | 0.99 |
| g7jac180 | 1.28 | 0.79 | 1.32 | 1.93 | 1.25 | 1.92 |
| mark3jac140sc | 1.40 | 2.73 | 1.45 | 1.47 | 1.24 | 1.48 |
| lhr71 | 1.71 | 0.98 | 1.69 | 2.85 | 2.73 | 2.84 |
| finan512 | 2.55 | 4.00 | 2.59 | 1.98 | 1.84 | 1.96 |
| nasasrb | 1.89 | 0.77 | 1.65 | 5.55 | 4.85 | 5.57 |
| lp_nug30 | 7.22 | 3.72 | 7.09 | 7.62 | 4.74 | 8.03 |
| cfd1 | 2.59 | 0.99 | 1.93 | 4.88 | 4.00 | 4.88 |
| shyy161 | 2.93 | 1.98 | 2.62 | 0.80 | 0.78 | 0.80 |
| venkat01 | 1.67 | 3.11 | 1.74 | 2.80 | 2.66 | 2.72 |
| qa8fk | 2.33 | 3.39 | 2.37 | 4.24 | 3.35 | 4.24 |
| pkustk03 | 4.10 | 3.12 | 4.10 | 6.41 | 5.72 | 6.42 |
| Average | 2.48 | 2.12 | 2.43 | 4.98 | 2.78 | 5.07 |

TABLE VIII
SOLUTION QUALITY FOR BOTH ALGORITHMS AND ALL THREE METRICS ON THE TEST SET FOR $K = 256$ AND $1\%$ LOAD IMBALANCE.

| Matrix | Sequential DP | | | PaToH | | |
|---|---|---|---|---|---|---|
| | Con-1 | Cut | SOED | Con-1 | Cut | SOED |
| epb3 | 102535 | 77414 | 179949 | 15520 | 14578 | 30651 |
| ct20stif | 107624 | 47013 | 154936 | 56416 | 34429 | 94277 |
| circuit_4 | 50071 | 36458 | 86539 | 38871 | 25406 | 64900 |
| bayer01 | 12671 | 12368 | 25047 | 8810 | 8704 | 17566 |
| gupta2 | 274309 | 61993 | 336373 | 203491 | 60797 | 266464 |
| bcircuit | 84097 | 52080 | 136267 | 7447 | 6666 | 14392 |
| g7jac180 | 254054 | 36664 | 290761 | 33285 | 18066 | 55928 |
| mark3jac | 74669 | 40128 | 114884 | 31399 | 20049 | 53824 |
| lhr71 | 26027 | 24864 | 50981 | 23686 | 22496 | 46998 |
| finan512 | 78340 | 50275 | 128669 | 38383 | 31050 | 70593 |
| nasasrb | 127422 | 54686 | 182113 | 52051 | 36705 | 90730 |
| lp_nug30 | 1137817 | 379350 | 1517167 | 651584 | 360049 | 1032354 |
| cfd1 | 221421 | 70641 | 292066 | 82151 | 48612 | 131200 |
| shyy161 | 67510 | 64015 | 131641 | 12572 | 11878 | 24611 |
| venkat01 | 129560 | 60220 | 189996 | 31400 | 26364 | 57856 |
| qa8fk | 171750 | 66127 | 237877 | 79579 | 44981 | 125009 |
| pkustk03 | 151848 | 60462 | 212880 | 48756 | 35076 | 86556 |
| Average | 180690 | 70280 | 251067 | 83259 | 47406 | 133171 |

IX clearly show that the DP approach is well suited for this type of problem, since the results table it builds does not need to account for $K$. It takes approximately the same time as it took for the 256-way partitioning, while PaToH slows down by about $50\%$. Due to the large number of partitions, the cutsizes increase dramatically, but again the gap between the algorithms closes, especially for the cut-net metric (see Table X).

### C. Instance dependent behaviour

To illustrate the determinants of the performance of the DP algorithm, we take a closer look at two matrices in our test set for which the performance of both algorithms differs considerably. The first such matrix is *nasasrb* in Figure 6(a) from the NASA (we use `cspy` to visualize the matrices [14]). Clearly, the fact that the interval spanned by each row is very short significantly reduces the necessary amount of computation for the DP algorithm. PaToH on the other hand does not benefit in the same way from this ordering.

TABLE IX
RUNNING TIME FOR BOTH ALGORITHMS AND ALL THREE METRICS WHEN CREATING VERY SMALL PARTITIONS IN MATRICES FROM THE TEST SET.

| Matrix | Sequential DP | | | PaToH | | |
|---|---|---|---|---|---|---|
| | Con-1 | Cut | SOED | Con-1 | Cut | SOED |
| epb3 | 2.77 | 2.10 | 2.82 | 3.46 | 3.11 | 3.47 |
| ct20stif | 1.54 | 0.73 | 1.55 | 8.27 | 5.98 | 8.32 |
| circuit_4 | 4.17 | 0.79 | 1.27 | 2.44 | 1.34 | 2.37 |
| bayer01 | 1.01 | 0.60 | 1.05 | 1.91 | 1.67 | 1.92 |
| gupta2 | 2.05 | 1.63 | 2.10 | 37.77 | 6.62 | 37.36 |
| bcircuit | 1.36 | 0.87 | 1.41 | 2.83 | 2.63 | 2.93 |
| g7jac180 | 1.67 | 0.57 | 1.11 | 3.58 | 2.14 | 3.59 |
| mark3jac | 1.19 | 1.56 | 1.26 | 3.90 | 2.82 | 3.94 |
| lhr71 | 1.54 | 0.78 | 1.62 | 5.37 | 4.48 | 5.38 |
| finan512 | 2.09 | 1.77 | 2.05 | 4.78 | 3.53 | 4.78 |
| nasasrb | 1.62 | 1.43 | 1.62 | 8.17 | 6.41 | 8.23 |
| lp_nug30 | 7.40 | 4.30 | 7.71 | 12.50 | 6.92 | 12.10 |
| cfd1 | 1.87 | 1.82 | 1.96 | 10.98 | 6.61 | 10.90 |
| shyy161 | 2.49 | 1.89 | 2.52 | 2.52 | 2.35 | 2.58 |
| venkat01 | 2.28 | 0.70 | 1.53 | 5.06 | 4.14 | 5.03 |
| qa8fk | 2.29 | 1.70 | 2.32 | 9.64 | 5.62 | 9.58 |
| pkustk03 | 2.59 | 1.65 | 2.66 | 8.99 | 7.24 | 8.95 |
| Average | 2.35 | 1.46 | 2.15 | 7.77 | 4.33 | 7.73 |

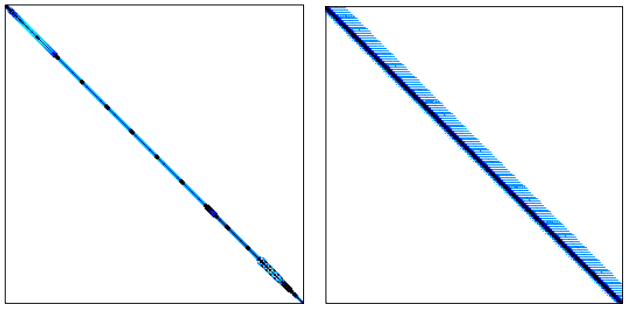
TABLE X
SOLUTION QUALITY FOR BOTH ALGORITHMS AND ALL THREE METRICS WHEN CREATING VERY SMALL PARTITIONS IN MATRICES FROM THE TEST SET.

| Matrix | Sequential DP | | | PaToH | | |
|---|---|---|---|---|---|---|
| | Con-1 | Cut | SOED | Con-1 | Cut | SOED |
| epb3 | 121888 | 84454 | 206342 | 32652 | 28666 | 62951 |
| ct20stif | 133359 | 49516 | 183254 | 90906 | 44358 | 137848 |
| circuit_4 | 61878 | 39592 | 101556 | 46750 | 29884 | 77553 |
| bayer01 | 18414 | 18390 | 36806 | 20910 | 15891 | 40134 |
| gupta2 | 451662 | 62064 | 513726 | 296184 | 61263 | 360490 |
| bcircuit | 87365 | 52998 | 140749 | 19578 | 16393 | 36762 |
| g7jac180 | 268582 | 38929 | 307512 | 48592 | 20713 | 77019 |
| mark3jac | 88746 | 39996 | 129036 | 48598 | 27375 | 80555 |
| lhr71 | 47866 | 44741 | 94339 | 57116 | 39630 | 99590 |
| finan512 | 107072 | 45568 | 152640 | 72163 | 39445 | 119419 |
| nasasrb | 143362 | 54800 | 198168 | 87645 | 49813 | 138423 |
| lp_nug30 | 1137929 | 379350 | 1517279 | 706750 | 368748 | 1076190 |
| cfd1 | 292898 | 70656 | 363554 | 147132 | 61173 | 210546 |
| shyy161 | 127934 | 76322 | 204256 | 25406 | 22831 | 47805 |
| venkat01 | 219096 | 62256 | 281384 | 60472 | 43148 | 103980 |
| qa8fk | 241076 | 66127 | 307203 | 134034 | 56282 | 192237 |
| pkustk03 | 227628 | 62394 | 290148 | 92148 | 51564 | 144468 |
| Average | 222162 | 73421 | 295762 | 116884 | 57481 | 176822 |

Opposed to this instance, the matrix *mark3jac140sc* from the Hollinger group (which is shortened to mark3jac in the tables) constitutes a difficult instance for the sequential DP approach. It is shown in Figure 6(b). Due to the fact that columns span much longer intervals here, the DP algorithm slows down significantly, compared to PaToH.

## V. CONCLUSION

We investigated one dimensional partitioning of sparse matrices under a given row ordering with the goal of optimizing well-known partitioning functions, including the total volume of communication in distribute memory sparse matrix vector multiply operations. This sequential partitioning problem is formulated using hypergraph partitioning models. We have developed cut-model graphs for three common metrics used



(a) *nasasrb* (b) *mark3jac140sc*

Fig. 6. Images of different matrices

in hypergraph partitioning. Unlike the general setting, the sequential partitioning problem allows for such cut models. We also developed an extended version of a known algorithm that deals with the requirements of a graph partitioning problem. Using the cut-models and the extended algorithm, we obtained a method for solving the sequential sparse matrix partitioning problem.

We have used a simple implementation of our algorithm in order to compare performance and solution quality to established general hypergraph partitioners. In these experiments, we have seen that the dynamic programming algorithm has a performance comparable to that of a hypergraph partitioner unless the number of partitions is low, in which case our approach is considerably slower, while the partition quality is always noticeably inferior in the sequential setting.

Thus, the proposed algorithm should not be used as a general hypergraph partitioner, but it constitutes a viable option for fixed row order partitioning problems. Furthermore, when the task is to partition the input into a large number of small parts, our algorithm outperforms PaToH by a large margin, making it also suitable for tasks of this type. In future work, we intend to improve the running time of our implementation. We also intend to study the effects of using row-ordering heuristics as initialization for our algorithm.

## REFERENCES


[1] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning: A survey," *Integration*, vol. 19, pp. 1–81, 1995.

[2] C. Alpert and A. Kahng, "Multiway partitioning via geometric embeddings, orderings, and dynamic programming," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 11, pp. 1342–1358, November 1995.

[3] R. Bellman, *Dynamic Programming*, 1st ed. Princeton, NJ, USA: Princeton University Press, 1957.

[4] R. H. Bisseling and I. Flesch, "Mondriaan sparse matrix partitioning for attacking cryptosystems by a parallel block Lanczos algorithm–A case study," *Parallel Comput.*, vol. 32, pp. 551–567, 2006.

[5] E. Boman, K. Devine, R. Heaphy, B. Hendrickson, V. Leung, L. A. Riesen, C. Vaughan, Ü. V. Çatalyürek, D. Bozdağ, W. Mitchell, and J. Teresco, *Zoltan 3.0: Parallel Partitioning, Load Balancing, and Data-Management Services; User's Guide*, Sandia National Laboratories, Albuquerque, NM, 2007, tech. Report SAND2007-4748W.

[6] A. Caldwell, A. Kahng, and I. Markov, "Improved algorithms for hypergraph bipartitioning," in *Design Automation Conference, 2000. Proceedings of the ASP-DAC 2000. Asia and South Pacific*, june 2000, pp. 661–666.

[7] Ü. V. Çatalyürek, C. Aykanat, and B. Uçar, "On two-dimensional sparse matrix partitioning: Models, methods, and a recipe," *SIAM J. Sci. Comput.*, vol. 32, pp. 656–683, 2010.

[8] Ü. V. Çatalyürek, C. Aykanat, and E. Kayaaslan, "Hypergraph partitioning-based fill-reducing ordering for symmetric matrices," *SIAM Journal on Scientific Computing*, vol. 33, no. 4, pp. 1996–2023, 2011.

[9] Ü. V. Çatalyürek, K. Kaya, and B. Uçar, "On shared-memory parallelization of a sparse matrix scaling algoritm," in *The 41st International Conference on Parallel Processing, ICPP*, 2012, accepted to be published.

[10] Ü. V. Çatalyürek, B. Uçar, and C. Aykanat, "Hypergraph partitioning," in *Encyclopedia of Parallel Computing*, D. A. Padua, Ed. Springer, 2011, pp. 871–881.

[11] U. V. Çatalyürek and C. Aykanat, "Decomposing irregularly sparse matrices for parallel matrix-vector multiplications," in *Proceedings of 3rd International Symposium on Solving Irregularly Structured Problems in Parallel, Irregular'96*, ser. Lecture Notes in Computer Science, no. 1117. Springer-Verlag, 1996, pp. 75–86.

[12] Ü. V. Çatalyürek and C. Aykanat, "Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication," *IEEE Transactions Parallel and Distributed Systems*, vol. 10, no. 7, pp. 673–693, 1999.

[13] ——, "PaToH: A multilevel hypergraph partitioning tool, version 3.0," Computer Engineering Department, Bilkent University, Tech. Rep. BU-CE-9915, 1999.

[14] T. A. Davis, *Direct Methods for Sparse Linear Systems*, ser. Fundamentals of Algorithms. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2006, no. 2.

[15] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1:1–1:25, 2011.

[16] E. Ihler, D. Wagner, and F. Wagner, "Modeling hypergraphs by graphs with the same mincut properties," *Information Processing Letters*, vol. 45, no. 4, pp. 171–175, 1993.

[17] G. Karypis and V. Kumar, *hMeTiS: A hypergraph partitioning package*, Minneapolis, MN 55455, 1998.

[18] ——, "Multilevel k-way hypergraph partitioning," *VLSI Des.*, vol. 11, pp. 285–300, 2000.

[19] B. W. Kernighan, "Optimal sequential partitions of graphs," *J. ACM*, vol. 18, no. 1, pp. 34–40, Jan. 1971.

[20] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*. Chichester, U.K.: Wiley–Teubner, 1990.

[21] M. Manguoglu, A. Sameh, and O. Schenk, "PSPIKE: Parallel sparse linear system solver," in *In Proc. Euro-Par 2009 Parallel Processing*, 2009, pp. 797–808.

[22] A. Pinar and C. Aykanat, "Sparse matrix decomposition with optimal load balancing," in *4-th International Conference on High Performance Computing*, Bangalore, India, December 1997, pp. 224–229.

[23] ——, "Fast optimal load balancing algorithms for 1d partitioning," *Journal of Parallel and Distributed Computing*, vol. 64, no. 8, pp. 974 – 996, 2004.

[24] M. Sathe, O. Schenk, B. Uçar, and A. Sameh, "A scalable hybrid linear solver based on combinatorial algorithms", in combinatorial scientific computing," in *Combinatorial Scientific Computing*, U. Naumann and O. Schenk, Eds. CRC Press, 2012, ch. 4, pp. 95–127.

[25] A. Trifunovic and W. Knottenbelt, "Parkway 2.0: A parallel multilevel hypergraph partitioning tool," in *Computer and Information Sciences - ISCIS 2004*, ser. Lecture Notes in Computer Science, C. Aykanat, T. Dayar, and I. Korpeoglu, Eds. Springer Berlin / Heidelberg, 2004, vol. 3280, pp. 789–800.

[26] B. Uçar, Ü. V. Çatalyürek, and C. Aykanat, "PaToH MATLAB interface," http://bmi.osu.edu/~umit/software.html, July 2009.

[27] B. Uçar, Ü. V. Çatalyürek, and C. Aykanat, "A matrix partitioning interface to PaToH in MATLAB," *Parallel Computing*, vol. 36, no. 5-6, pp. 254–272, 2010.

[28] B. Vastenhouw and R. H. Bisseling, "A two-dimensional data distribution method for parallel sparse matrix-vector multiplication," *SIAM Review*, vol. 47, no. 1, pp. 67–95, 2005.

[29] I. Yamazaki, X. S. Li, F.-H. Rouet, and B. Uçar, "Combinatorial problems in a parallel hybrid linear solver," Dept. Comp. Sci., RWTH Aachen Univ., Tech. Rep. AIB 2011-09, 2011.